# Section exam 3 Numerical Analysis: Linear Algebra

**Solutions**

Assigned Nov 21, 2024. Due Wednesday Dec 4, 2024, at noon.

**Instructions:**

- ***This is an open book exam.***
  You are allowed to consult the text book. You are also allowed to do research in other text books, on the internet, etc.
- The exam should be worked ***individually***. Do not consult with other students.
- Please type your solutions if at all possible.
- Some of the questions involve writing Matlab code. If at all possible, please stick to Matlab. However, if you absolutely insist, you can use other similar languages, as long as the syntax is sufficiently similar that your code is readable to somebody who does not know the language. If you use a different language, adding comments describing what each line does would be helpful.
- The following files are provided on Canvas and the course webpage:
  - `exam3_student_version.m` provides starting points for problems 4 and 5.
  - `matrix_Y.mat` holds the matrix $\mathbf{Y}$ that you need in problem 4 in Matlab format.
  - `matrix_Y.txt` holds the matrix $\mathbf{Y}$ that you need in problem 4 in plain text format.

**Question 1:** (20p) Let $\mathbf{A}$ be an invertible sparse matrix of size $m \times m$ with the singular value decomposition

$$\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{V}^*.$$

Further, let $\mathbf{b}$ be a vector of size $m \times 1$, and suppose that you seek to solve the linear system

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{1}$$

for $\mathbf{x}$. Suppose that $m$ is large enough that you cannot use dense methods, but that you can afford to apply $\mathbf{A}$ and $\mathbf{A}^*$ to vectors.

(a) (7 points) Since $\mathbf{A}$ is invertible, the system (1) is mathematically equivalent to the system

$$\mathbf{A}^*\mathbf{A}\mathbf{x} = \mathbf{A}^*\mathbf{b}. \tag{2}$$

Would you be able to use Conjugate Gradients to solve (2)? If you answer yes, then discuss the costs, and the pros and cons of this approach; identify some situations where this approach may work well, and some where it may not work well. If you answer no, then describe why CG cannot be used.

*Yes, when $\mathbf{A}$ is non-singular, the matrix $\mathbf{A}^*\mathbf{A}$ is both symmetric and positive definite, so CG can be used. The cost per iteration is higher than, e.g., GMRES in that you need **two** matvecs instead of one. However, due to the three term recursion, the number of flops required outside of the matvec is less. CG also requires less memory.*

*The key drawback of using CG to solve the normal equations is that you **square** the condition number of the matrix. For an ill-conditioned matrix, this will likely slow down the speed of convergence, and may lead to serious loss of accuracy due to round-off errors.*

*To summarize, the approach outlined would work well for well-conditioned problems, in particular in cases where the spectrum of $\mathbf{A}^*\mathbf{A}$ is clustered. However, this approach should not be used for highly ill-conditioned problems.*

(b) (6 points) Consider the matrix

$$\mathbf{B} = \begin{bmatrix} \mathbf{0} & \mathbf{A} \\ \mathbf{A}^* & \mathbf{0} \end{bmatrix}.$$

Let $\mathbf{u}_i$ and $\mathbf{v}_i$ be a pair of left and right singular vectors of $\mathbf{A}$, so that $\mathbf{A}\mathbf{v}_i = \sigma_i\mathbf{u}_i$. Prove that the vectors

$$\begin{bmatrix} \mathbf{u}_i \\ \mathbf{v}_i \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} \mathbf{u}_i \\ -\mathbf{v}_i \end{bmatrix}$$

are both eigenvectors of $\mathbf{B}$, and then provide the full eigenvalue decomposition of $\mathbf{B}$, expressed in terms of the matrices $\mathbf{U}$, $\mathbf{D}$, and $\mathbf{V}$.

*A simple computation shows that*

$$\begin{bmatrix} \mathbf{0} & \mathbf{A} \\ \mathbf{A}^* & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{u}_i \\ \mathbf{v}_i \end{bmatrix} = \begin{bmatrix} \mathbf{A}\mathbf{v}_i \\ \mathbf{A}^*\mathbf{u}_i \end{bmatrix} = \begin{bmatrix} \sigma_i\mathbf{u}_i \\ \sigma_i\mathbf{v}_i \end{bmatrix} = \sigma_i \begin{bmatrix} \mathbf{u}_i \\ \mathbf{v}_i \end{bmatrix}.$$

*Analogously*

$$\begin{bmatrix} \mathbf{0} & \mathbf{A} \\ \mathbf{A}^* & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{u}_i \\ -\mathbf{v}_i \end{bmatrix} = \begin{bmatrix} -\mathbf{A}\mathbf{v}_i \\ \mathbf{A}^*\mathbf{u}_i \end{bmatrix} = \begin{bmatrix} -\sigma_i\mathbf{u}_i \\ \sigma_i\mathbf{v}_i \end{bmatrix} = -\sigma_i \begin{bmatrix} \mathbf{u}_i \\ (-\mathbf{v}_i) \end{bmatrix}.$$

*It follows that the columns of the matrix*

$$\mathbf{W} = \frac{1}{\sqrt{2}} \begin{bmatrix} \mathbf{U} & \mathbf{U} \\ \mathbf{V} & -\mathbf{V} \end{bmatrix}$$

*form an orthonormal set of eigenvectors of $\mathbf{B}$, and that*

$$\mathbf{B} = \mathbf{W} \begin{bmatrix} \mathbf{D} & \mathbf{0} \\ \mathbf{0} & -\mathbf{D} \end{bmatrix} \mathbf{W}^*$$

*is an eigenvalue decomposition of $\mathbf{B}$.*

(c) (7 points) In order to solve (1), you could in principle instead solve the linear system

$$\begin{bmatrix} \mathbf{0} & \mathbf{A} \\ \mathbf{A}^* & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{y} \\ \mathbf{x} \end{bmatrix} = \begin{bmatrix} \mathbf{b} \\ \mathbf{0} \end{bmatrix}. \tag{3}$$

Observe that the coefficient matrix in (3) is symmetric. Would you be able to use Conjugate Gradients to solve (3)? If you answer yes, then discuss the costs, and the pros and cons of this approach; identify some situations where this approach may work well, and some where it may not work well. If you answer no, then describe why CG cannot be used.

*No, this will not work. Since* $\mathbf{B}$ *has negative eigenvalues, it is not positive definite.*

**Question 2:** (20 points)

(a) (6 points) Consider a symmetric positive definite (spd) matrix $\mathbf{A}$ that has been partitioned as

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{12}^* & \mathbf{A}_{22} \end{bmatrix}.$$

Suppose that $\mathbf{A}_{11}$ has the Cholesky factorization

$$\mathbf{A}_{11} = \mathbf{R}_{11}^* \mathbf{R}_{11}.$$

Then $\mathbf{A}$ admits the partial factorization

$$\mathbf{A} = \mathbf{R}_1^* \mathbf{A}_1 \mathbf{R}_1$$

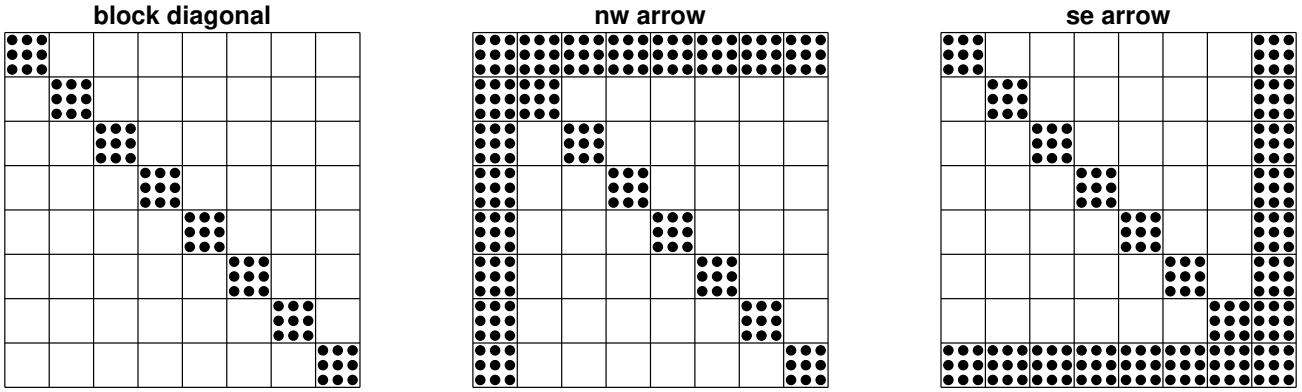
where

$$\mathbf{R}_1 = \begin{bmatrix} \mathbf{R}_{11} & \mathbf{R}_{12} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \qquad \text{and} \qquad \mathbf{A}_1 = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}'_{22} \end{bmatrix}.$$

Give formulas for $\mathbf{R}_{12}$ and $\mathbf{A}'_{22}$.

(b) (7 points) Suppose that $\mathbf{A}$ is an spd matrix $\mathbf{A}$ of size $N \times N$, where $N = nb$ for two positive integers $n$ and $b$. Write a Matlab code that computes the Cholesky factorization of $\mathbf{A}$ that is *blocked* with a block size $b$. (In other words, it will consist of a loop with $n-1$ steps, where in each step a rank $b$ update is applied to the bottom right block of the matrix, and a final step where the last remining block is factorized.) Submit a printout of the code as your answer to this question. You are allowed to invoke the built-in command `chol`, but only to matrices of size $b \times b$.

(c) (7 points) Consider the Cholesky factorization of matrices with the following block patterns:




*In the examples,* $b = 3$ *and* $n = 8$.

Suppose that you write a code that exploits the zero structure of the matrices, as far as is possible in each case. What would the asymptotic flop count be for each of the patterns, expressed as a function of $n$ and $b$? To illustrate, for a general dense matrix, your answer would be $\boxed{O(n^3 b^3)}$

***Solution:***

(a)

$$\mathbf{R}_{12} = \left(\mathbf{R}_{11}^*\right)^{-1}\mathbf{A}_{12} \qquad \text{and} \qquad \mathbf{A}'_{22} = \mathbf{A}_{22} - \mathbf{R}_{12}^*\mathbf{R}_{12} = \mathbf{A}_{22} - \mathbf{A}_{12}^*\mathbf{A}_{11}^{-1}\mathbf{A}_{12}.$$

(Either formula for $\mathbf{A}'_{22}$ is fine, although you want to use $\mathbf{A}'_{22} = \mathbf{A}_{22} - \mathbf{R}_{12}^*\mathbf{R}_{12}$ in your code.)

(b) For instance:

```
R = A;
for istep = 1:(nblock-1)
  I      = (istep-1)*b + (1:b);
  J      = (istep*b+1):n;
  Rloc   = chol(R(I,I));
  R(I,I) = Rloc;
  R(I,J) = Rloc'\R(I,J);
  R(J,I) = zeros(length(J),length(I));
  R(J,J) = R(J,J) - R(I,J)'*R(I,J);
end
I      = (nblock-1)*b + (1:b);
Rloc   = chol(R(I,I));
R(I,I) = Rloc;
```

(c)

***block diagonal:*** Each diagonal block is factored independently of the others. The cost for one block is $O(b^3)$, and there are $n$ blocks so:

$$O(nb^3)$$

***nw arrow:*** After the first step, you are left with a matrix of size $(n-1)b \times (n-1)b$ that is fully dense in the lower right corner. Since there is no zero structure left to exploit, the total cost becomes

$$O(n^3b^3)$$

***se arrow:*** Each step involves factoring the diagonal block, then updating precisely three blocks of size $b \times b$. In consequence, the cost per step is just $O(b^3)$, and so

$$O(nb^3)$$

**Question 3:** (20 points) Let $\mathbf{A}$ be an $m \times m$ matrix that is symmetric and positive definite. Let $k$ be a positive integer that is smaller than $m$, and suppose that $\mathbf{G}$ is a matrix of size $m \times k$. If the matrix $\mathbf{G}^*\mathbf{A}\mathbf{G}$ is invertible, then we define the *Nyström* approximation to $\mathbf{A}$ (with respect to $\mathbf{G}$) via

$$\mathbf{B} := (\mathbf{A}\mathbf{G})\,(\mathbf{G}^*\mathbf{A}\mathbf{G})^{-1}\,(\mathbf{A}\mathbf{G})^*.$$

Let $\{\lambda_j\}_{j=1}^m$ denote the eigenvalues of $\mathbf{A}$, ordered by modulus so that

$$|\lambda_1| \geq |\lambda_2| \geq |\lambda_3| \geq \cdots \geq |\lambda_m|.$$

(a) (5 points) Suppose that $\mathbf{A}$ has exact rank $k$, and that it has the (economy size) eigenvalue decomposition

$$\underset{m \times m}{\mathbf{A}} = \underset{m \times k}{\mathbf{U}}\ \underset{k \times k}{\mathbf{D}}\ \underset{k \times m}{\mathbf{U}^*}.$$

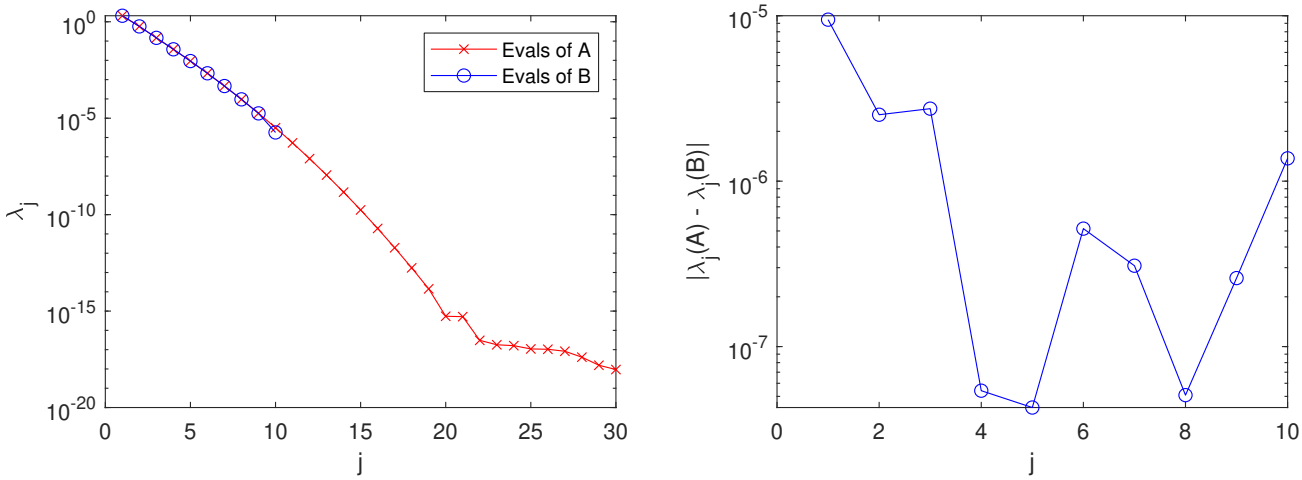
Prove that if the matrix $\mathbf{Z} := \mathbf{U}^*\mathbf{G}$ is non-singular, then the Nyström approximation is *exact*. In other words, $\mathbf{B} = \mathbf{A}$.

(b) (10 points) In parts (b) and (c), we will draw $\mathbf{G}$ from a Gaussian random distribution. In Matlab, this is done through the command `G = randn(m,k)`. It turns out that in this case, the Nyström approximation is often a very accurate approximation to $\mathbf{A}$ if the eigenvalues $\{\lambda_j\}_{j=k+1}^m$ are small in comparison to the top ones. In particular, the eigenvalues of $\mathbf{B}$ are very good approximations to the top $k$ eigenvalues of $\mathbf{A}$.

For instance, consider the $30 \times 30$ matrix $\mathbf{A}$ with entries

$$\mathbf{A}(i,j) = \frac{1}{\sqrt{i+j}}.$$

The left figure below plots the eigenvalues of $\mathbf{A}$ versus the eigenvalues of $\mathbf{B}$ for a particular instantiation of the method. The errors are shown to the right.




Code up the Nyström method, and replicate the graph! (Note that your graph may not look *exactly* like the one shown – the output depends on the draw of the random numbers!) Hand in your code and the graph.

(c) (5 points) Play with some other spd matrices whose eigenvalues decay. For your solution to (c), hand in descriptions of two matrices, and the plots analogous to those you created in (b).

*Note: For problem (c), fun examples of matrices that arise in real applications are appreciated! However, you get full credit as long as you show that you tested a couple of different things.*

***Solution:***

(a) Observe that

$$\mathbf{AG} = \mathbf{UDU}^*\mathbf{G} = \mathbf{UDZ},$$

and that

$$\mathbf{G}^*\mathbf{AG} = \mathbf{G}^*\mathbf{UDU}^*\mathbf{G} = \mathbf{Z}^*\mathbf{DZ}.$$

It follows that

$$\mathbf{B} = (\mathbf{UDZ})\left(\mathbf{Z}^*\mathbf{DZ}\right)^{-1}(\mathbf{UDZ})^* = (\mathbf{UDZ})\left(\mathbf{Z}^{-1}\mathbf{D}^{-1}\mathbf{Z}^{-*}\right)(\mathbf{Z}^*\mathbf{DU}^*) = \mathbf{UDU}^* = \mathbf{A}.$$

(b) For instance:

```
%%% Build the matrix A and compute its eigenvalues.
%%% Then sort them by magnitude.
[ii,jj] = meshgrid(1:n);
A       = 1./sqrt(ii.*ii + jj.*jj);
ee      = eig(A);
[~,ind] = sort(-abs(ee));
ee      = ee(ind);

%%% Build the matrix B and compute its eigenvalues.
%%% Then sort them by magnitude.
G       = randn(n,k);
B       = (A*G)*inv(G'*A*G)*(A*G)';
ff      = eig(B);
[~,ind] = sort(-abs(ff));
ff      = ff(ind);

%%% Plot the results
subplot(1,2,1)
semilogy(1:n,abs(ee(1:n)),'rx-',1:k,ff(1:k),'bo-')
legend('Evals of A','Evals of B')
subplot(1,2,2)
semilogy(1:k,abs(ee(1:k)-ff(1:k)),'bo-')
```

*Note: The code actually plots the* ***moduli*** *of the computed eigenvalues of* $\mathbf{A}$*. The reason is that some of the computed eigenvalues of size machine epsilon turn out to be negative. This is purely a numerical artifact – think of these as all* $\epsilon_{\text{mach}}$*.*

**Question 4:** (20 points) In this problem, you are tasked with determining approximations to some eigenvalues and eigenvectors of a real symmetric matrix $\mathbf{A}$ of size $100 \times 100$. In preparing this problem, I generated a random starting vector $\mathbf{q}$, and then executed 49 steps of power iteration to build the $100 \times 50$ matrix

$$\mathbf{Y} = \big[\mathbf{q},\, \mathbf{Aq},\, \mathbf{A}^2\mathbf{q},\, \mathbf{A}^3\mathbf{q},\, \ldots,\, \mathbf{A}^{49}\mathbf{q}\big].$$

You will find the resulting matrix $\mathbf{Y}$ ready for download on Canvas (and on the course webpage).

In this problem, the eigenvalues of $\mathbf{A}$ are ordered by modulus, so that

$$|\lambda_1| \geq |\lambda_2| \geq |\lambda_3| \geq \cdots \geq |\lambda_{100}|.$$

Using the information in $\mathbf{Y}$, compute approximations to the following quantities:

(a) (8 points) The largest (in modulus) eigenvalue $\lambda_1$.

(b) (7 points) The first five entries of the eigenvector $\mathbf{v}_1$ associated with $\lambda_1$. To resolve the normalization ambiguity, suppose that $\|\mathbf{v}_1\| = 1$ and that the first entry of $\mathbf{v}_1$ is positive.

(c) (5 points) The eigenvalue $\lambda_5$.

***Solutions:*** The precise answers are:

$$\begin{aligned}
\lambda_1 &= 1.050000000000000 \\
\lambda_5 &= 0.571081383523783 \\
\mathbf{v}_1(1) &= 0.703819618759918 \\
\mathbf{v}_1(2) &= 0.190186946209695 \\
\mathbf{v}_1(3) &= -0.123611936151842 \\
\mathbf{v}_1(4) &= 0.062117413353951 \\
\mathbf{v}_1(5) &= 0.056202439003498
\end{aligned}$$

There are several different ways to estimate these quantities using only information in $\mathbf{Y}$. Let us describe three of them, in increasing order of sophistication. We let $\mathbf{y}_j = \mathbf{A}^{j-1}\mathbf{q}$ denote the $j$'th column of $\mathbf{y}$.

**(1) Basic power iteration:** Simply use that $\frac{1}{|\mathbf{y}_j|} \to \pm\mathbf{v}_1$, and that for $k$ large, we expect $\mathbf{y}_{k+1} \approx \lambda_1 \mathbf{y}_k$. For instance, one may use

$$\lambda_1^{\text{approx}} = \frac{Y(1,50)}{Y(1,49)} = 0.86.$$

To estimate $\mathbf{v}_1$, just normalize $\mathbf{y}_{50}$ to get $\mathbf{v}_1^{\text{approx}} = \frac{\text{sign}(\mathbf{y}_{50}(1))}{\|\mathbf{y}_{50}\|}\mathbf{y}_{50} = \begin{bmatrix} 0.632895887827886 \\ 0.206732040476769 \\ -0.144706386514006 \\ 0.064704328677310 \\ 0.054143568598003 \\ \vdots \end{bmatrix}$

**(2) Power iteration with Rayleigh quotients:** To enhance the accuracy in estimating $\lambda_1$, let us use a Rayleigh quotient:

$$\lambda_1^{\text{approx}} = \frac{\mathbf{y}_k^* \mathbf{A}\mathbf{y}_k}{\mathbf{y}_k^*\mathbf{y}_k} = \frac{\mathbf{y}_k^*\mathbf{y}_{k+1}}{|\mathbf{y}_k|^2}.$$

Setting $k = 49$, we find

$$\lambda_1^{\text{approx}} = 1.026454737166689.$$

To get an idea of how many correct digits we have, we compute the last several quotients:

```
Raleigh quotient at step  45 =  1.021690860323
Raleigh quotient at step  46 =  1.022964240229
Raleigh quotient at step  47 =  1.024181064223
Raleigh quotient at step  48 =  1.025343780572
Raleigh quotient at step  49 =  1.026454737167
```

This is slightly hard to interpret, but one may perhaps guess that we have about one or two percent accuracy.

**(3) Block Rayleigh quotients:** The most sophisticated idea would be to use a *block Raleigh quotient*, as in the Lanzcos method. Let $\mathbf{Q}$ be an orthonormal basis for the first 49 columns of $\mathbf{Y}$. For instance, compute the QR factorization

$$\mathbf{Y}(:, 1:49) = \mathbf{QR}$$

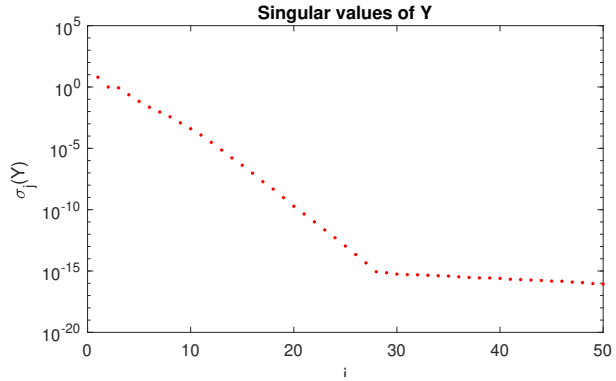
Then we seek to build the matrix

$$\mathbf{B} = \mathbf{Q}^*\mathbf{AQ},$$

with the idea that the eigenvalues of $\mathbf{B}$ are good approximations to the eigenvalues of $\mathbf{A}$. Observe that

$$\mathbf{B} = \mathbf{Q}^*\mathbf{AQ} = \mathbf{Q}^*\mathbf{AQRR}^{-1} = \mathbf{Q}^*\mathbf{AY}(:, 1:49)\mathbf{R}^{-1}. \tag{4}$$

The formula (4) is correct mathematically, but if you code it up, you will get junk! The reason is that $\mathbf{Y}$ numerically has much lower rank that 49, so the matrix $\mathbf{R}$ is singular.




The graph indicates that the numerical rank is around 25, so we try instead to work with the first 20 columns of $\mathbf{Y}$. To be precise, we execute the code:

```
p     = 20;
[Q,R] = qr(Y(:,1:p),0);
B     = Q'*Y(:,2:(p+1))*inv(R);
```

We now get excellent estimates:

$$\begin{aligned}\lambda_1^{\text{approx}} &= \lambda_1(\mathbf{B}) = 1.049999999999994\\ \lambda_5^{\text{approx}} &= \lambda_5(\mathbf{B}) = 0.571081184207039\end{aligned}$$

For question (b), we compute the dominant eigenvector of $\mathbf{QQ}^*\mathbf{AQQ}^*$, and find

$$\mathbf{v}_1^{\text{approx}} = \begin{bmatrix} 0.703819618755760\\ 0.190186946212856\\ -0.123611936105692\\ 0.062117413348342\\ 0.056202438977621\\ \vdots \end{bmatrix}$$

(It turns out that $\|\mathbf{v}_1 - \mathbf{v}_1^{\text{approx}}\|_\infty \approx 6 \cdot 10^{-11}$, which is pretty good!)

*Notes:* Method (3) is quite sophisticated – kudos to anyone who thought of it! Fwiw, I do not know of a simple way to answer (c).

**Question 5:** (20 points) On Canvas (and on the course webpage) you will find a Matlab code that executes the Jacobi method that we discussed in class. This is a method that takes a given symmetric matrix $\mathbf{A}$ and iteratively drives it to diagonal form through a series of similarity transforms. To be precise, recall from class that for any off-diagonal entry $(i, j)$ of $\mathbf{A}$, it is possible to find a Givens rotation $\mathbf{G}$ that maps the submatrix $\mathbf{A}([i, j], [i, j])$ to diagonal form:

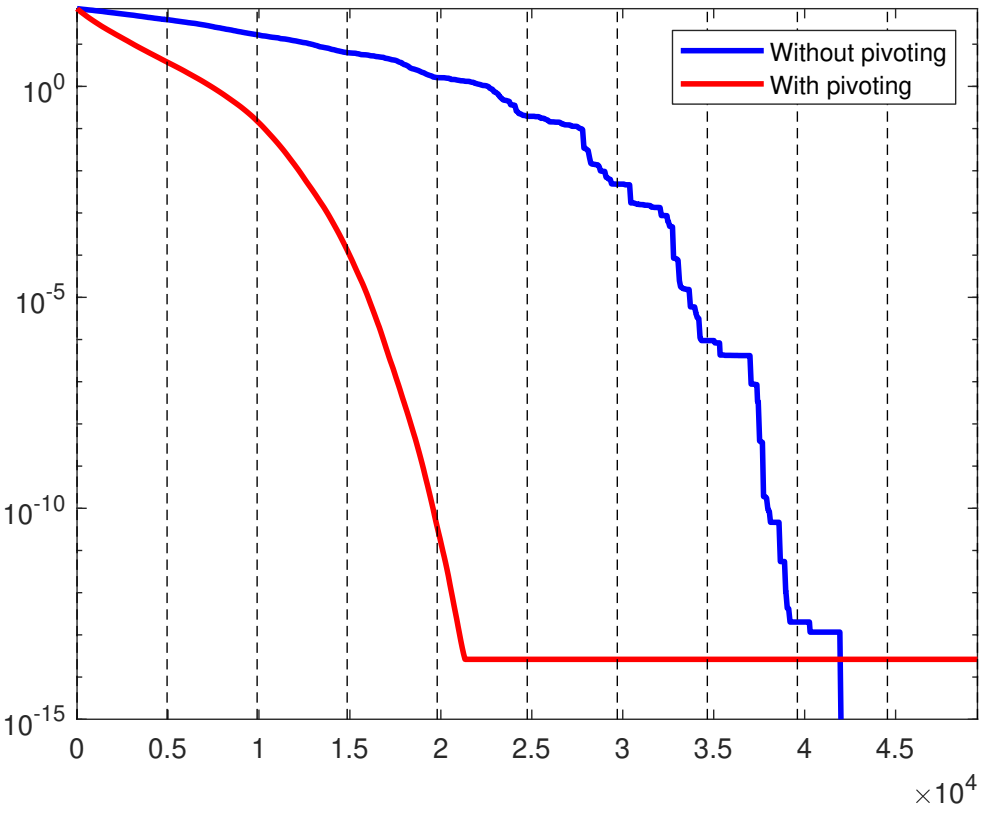
$$\mathbf{G}^* \begin{bmatrix} a_{i,i} & a_{i,j} \\ a_{i,j} & a_{j,j} \end{bmatrix} \mathbf{G} = \begin{bmatrix} a'_{i,i} & 0 \\ 0 & a'_{j,j} \end{bmatrix} \tag{5}$$

In the Jacobi iteration, you repeatedly loop over all the off-diagonal entries of $\mathbf{A}$. When entry $(i, j)$ is processed, the method constructs a Givens rotation $\mathbf{G}$ that drives the $(i, j)$ and $(j, i)$ entries to zero through a similarity transform where $\mathbf{G}^*$ acts on the $(i, j)$ rows from the left and $\mathbf{G}$ on the $(i, j)$ columns from the right.

In the code provided in the file `exam3_student_version.m`, the off-diagonal entries are processed in a fixed order, sweeping through one row of the matrix at a time.

To accelerate convergence, one may instead conduct a search to find the off-diagonal entry with the largest modulus, and target it instead.

The plot below shows the Frobenius norm of the off-diagonal part of the matrix after step $j$ of the iteration. The blue line shows the error from the unpivoted method, and the red line shows the error from the pivoted method.




Write a code that implements the pivoted version of the Jacobi iteration. Submit an error plot analogous to the one shown, as well as a printout of your code.

*Note: The pivoted version of Jacobi is far too expensive to use in practice. It increases the cost of one step of the method from $O(m)$ to $O(m^2)$! Efficiency would be greatly improved if you first drive $\mathbf{A}$ to tridiagonal form, and then exploit the tridiagonal structure.*

***Solution:*** For instance:

```
%%% Without pivoting.
T = A;
ff = zeros(1,niter);
ndone = 1;
for k = 1:npass
  for i = 1:(m-1)
    for j = (i+1):m
      beta = (T(j,j) - T(i,i))/(2*T(i,j));
      t    = sign(beta)/(abs(beta) + sqrt(beta*beta+1));
      c    = 1/sqrt(t*t+1);
      J    = [c,c*t;-c*t,c];
      T([i,j],:) = J'*T([i,j],:);
      T(:,[i,j]) = T(:,[i,j])*J;
      ff(ndone) = sqrt(sum(sum(triu(T,1).^2)));
      ndone = ndone + 1;
    end
  end
end

%%% With pivoting.
T = A;
ee = zeros(1,niter);
for k = 1:niter
  B = abs(T - diag(diag(T)));
  [mm,jj] = max(B);
  [~,i] = max(mm);
  j = jj(i);
  beta = (T(j,j) - T(i,i))/(2*T(i,j));
  t    = sign(beta)/(abs(beta) + sqrt(beta*beta+1));
  c    = 1/sqrt(t*t+1);
  J    = [c,c*t;-c*t,c];
  T([i,j],:) = J'*T([i,j],:);
  T(:,[i,j]) = T(:,[i,j])*J;
  ee(k) = sqrt(sum(sum(triu(T,1).^2)));
end
```